Terraform Task

Name:Shaik Khaja Basha
Batch : Batch 11
Date : 25.07.2025
Task : for expression and functions in Terraform

Terraform for Expressions:

Terraform's for expressions allow you to transform collections (lists, sets, maps) in powerful ways. Here's a comprehensive guide with examples:

Basic Syntax

List Iteration

[for item in LIST : EXPRESSION]

- Returns a new list
- item represents each element

Map Iteration

[for key, value in MAP : EXPRESSION]

{for key, value in MAP : KEY_EXPRESSION => VALUE_EXPRESSION}

- Can return either a list or new map

Practical Examples

A. Transforming Lists

```
variable "filename" {
  type    = list(string)
  default = ["a", "b", "c"]
}
resource "local_file" "f8" {
  count    = length(var.filename_upper)
  filename = var.filename_upper[count.index]
  content  = "test"
}
```

Working with Maps

```
variable filnamemap {
 type    = map(string)
 default = {
          name ="a"
          address = "b"
          }
}
```

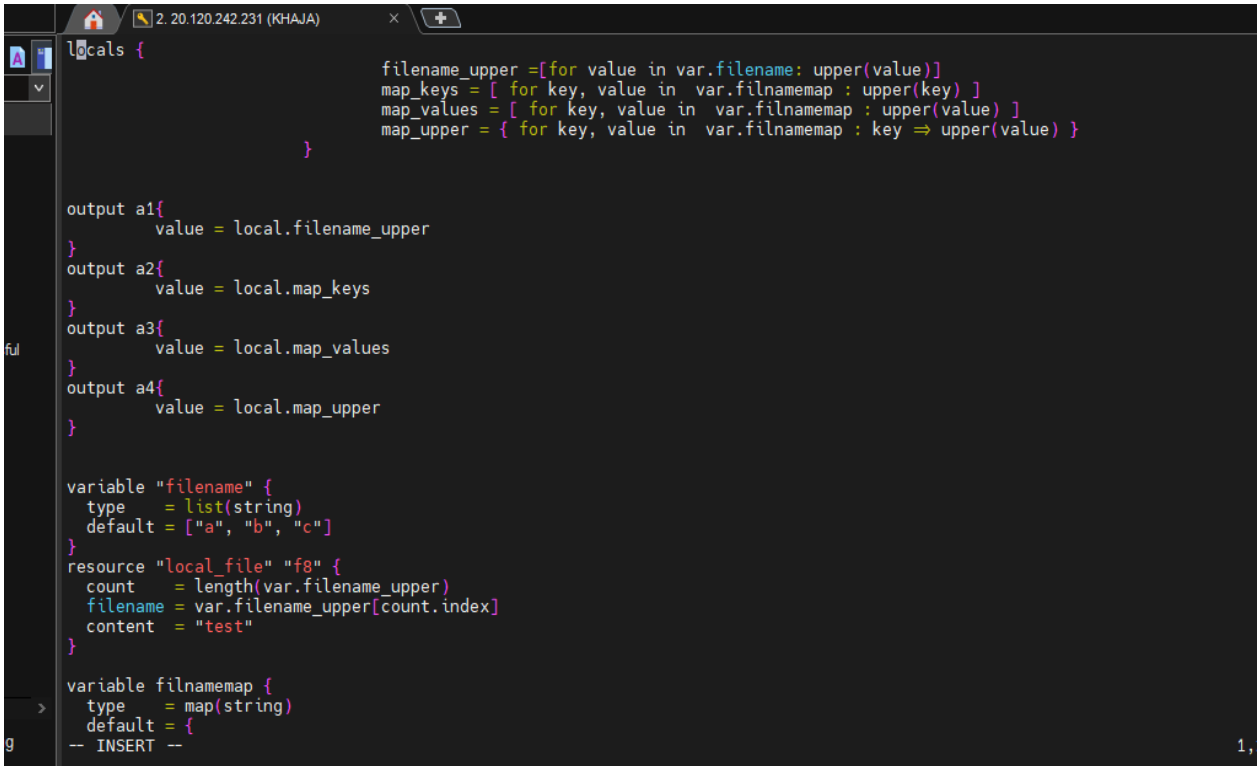
```
["name","address"]
```

```
["a",'b"]
```

Create a file name for the configuration as res.tf and put the values save and quit ess:wq

```
locals {
                    filename_upper =[for value in var.filename: upper(value)]
                    map_keys = [ for key, value in  var.filnamemap : upper(key) ]
                    map_values = [ for key, value in  var.filnamemap : upper(value) ]
                    map_upper = { for key, value in  var.filnamemap : key => upper(value) }
                              }
output a1{
          value = local.filename_upper
}
output a2{
          value = local.map_keys
}
output a3{
          value = local.map_values
}
output a4{
          value = local.map_upper
}
```

```
variable "filename" {
  type    = list(string)
  default = ["a", "b", "c"]
}
resource "local_file" "f1" {
  count    = length(var.filename_upper)
  filename = var.filename_upper[count.index]
  content  = "test"
}
variable filnamemap {
  type    = map(string)
  default = {
          name ="a"
          address = "b"
          }
}
```



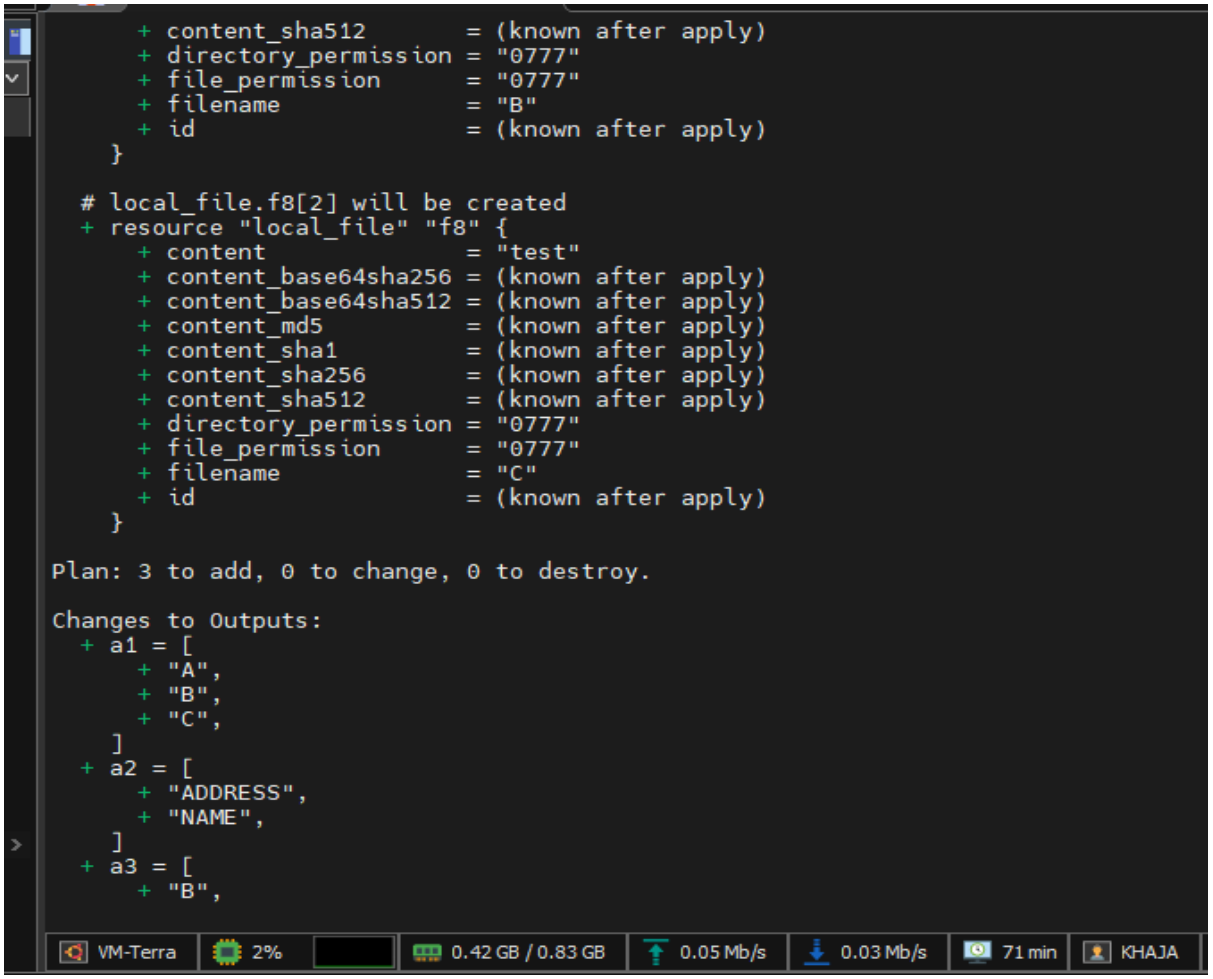
```
variable filnamemap {
  type     = map(string)
  default = {
         name ="a"
         address  = "b"
         }
}
-- INSERT --
```

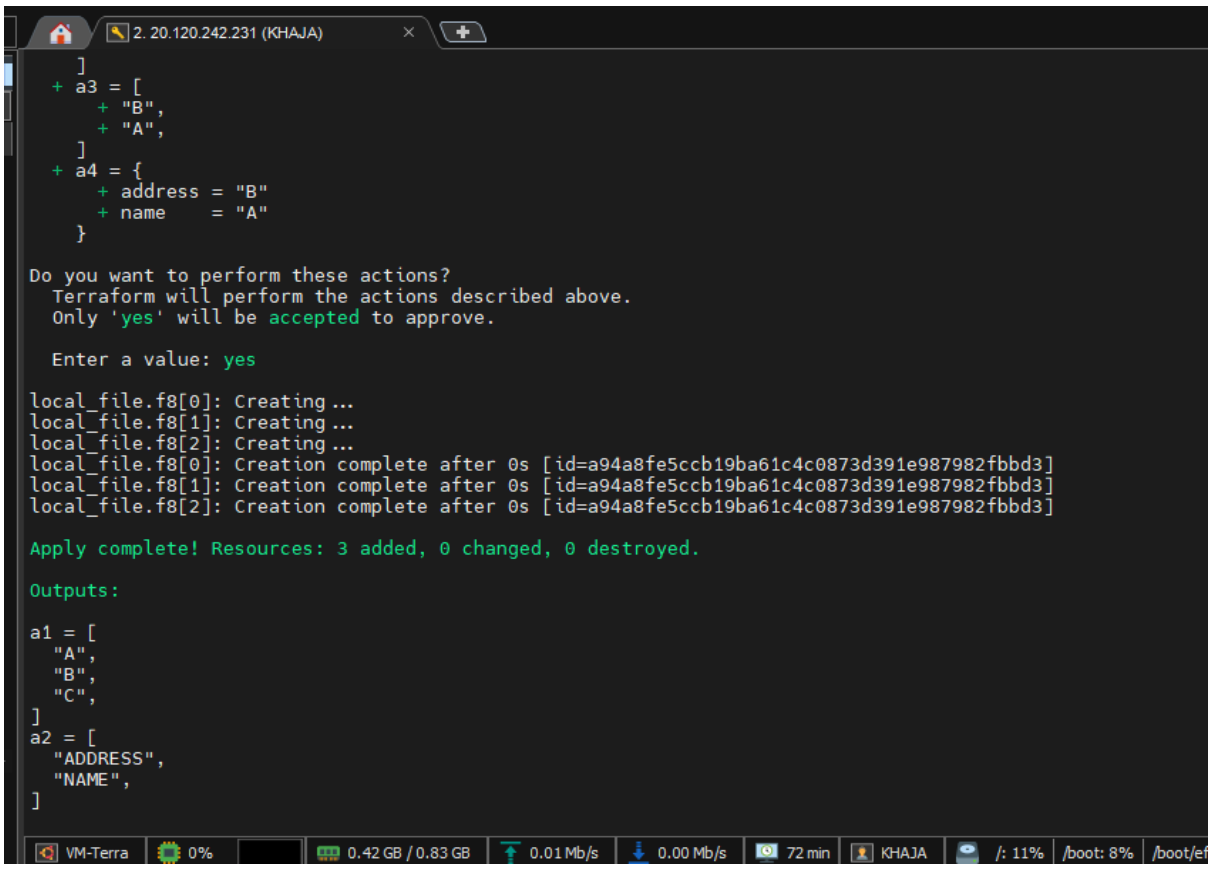
Execute the command **terraform init** for initializing the configuration

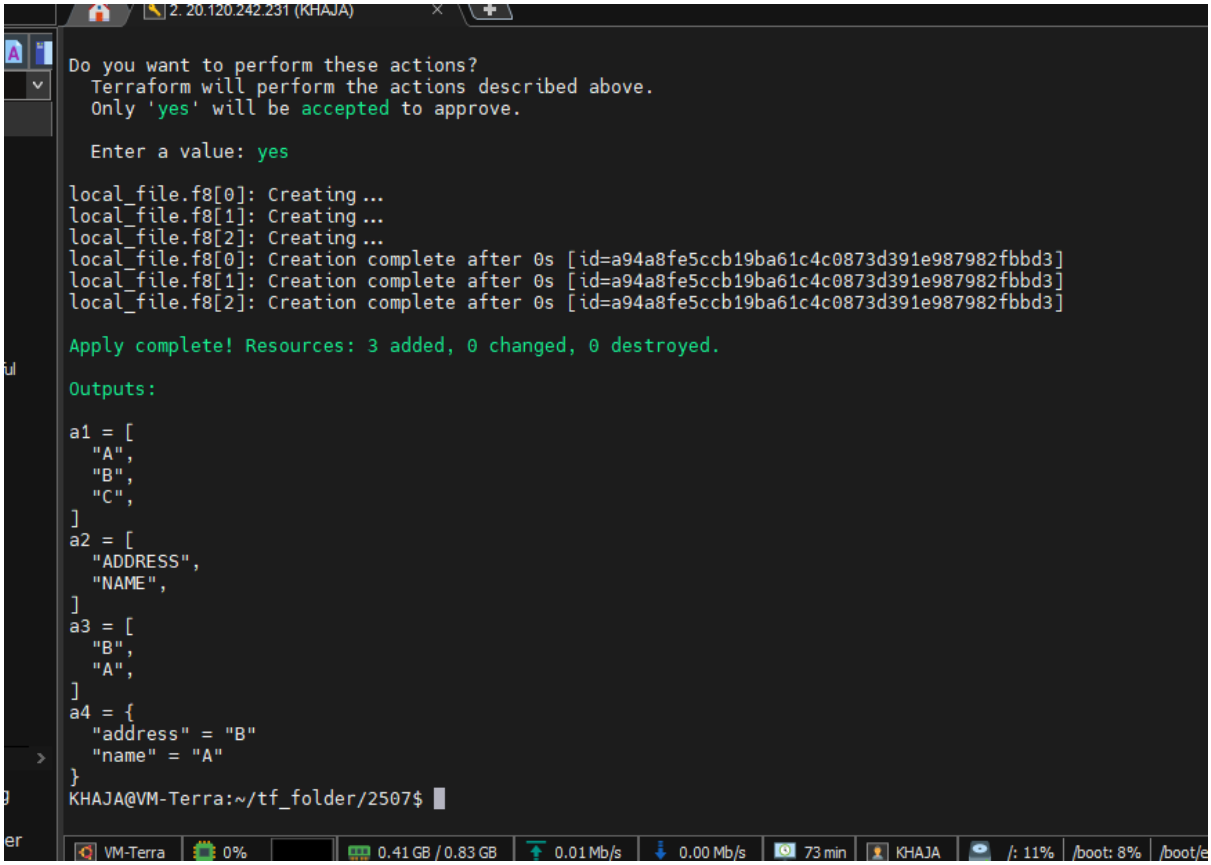
```
KHAJA@VM-Terra:~/tf_folder/2507$ vi res.tf
KHAJA@VM-Terra:~/tf_folder/2507$ terraform fmt
res.tf
KHAJA@VM-Terra:~/tf_folder/2507$ terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/local...
- Installing hashicorp/local v2.5.3...
- Installed hashicorp/local v2.5.3 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
KHAJA@VM-Terra:~/tf_folder/2507$
```

Run the command **terraform apply** to make the changes

```
KHAJA@VM-Terra:~/tf_folder/2507$ vi res.tf
KHAJA@VM-Terra:~/tf_folder/2507$ terraform apply

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following
symbols:
  + create

Terraform will perform the following actions:

  # local_file.f8[0] will be created
  + resource "local_file" "f8" {
      + content              = "test"
      + content_base64sha256 = (known after apply)
      + content_base64sha512 = (known after apply)
      + content_md5          = (known after apply)
      + content_sha1         = (known after apply)
      + content_sha256       = (known after apply)
      + content_sha512       = (known after apply)
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "A"
      + id                   = (known after apply)
    }

  # local_file.f8[1] will be created
  + resource "local_file" "f8" {
      + content              = "test"
      + content_base64sha256 = (known after apply)
      + content_base64sha512 = (known after apply)
      + content_md5          = (known after apply)
      + content_sha1         = (known after apply)
      + content_sha256       = (known after apply)
      + content_sha512       = (known after apply)
      + directory_permission = "0777"
      + file_permission      = "0777"
```


+ content_sha512 = (known after apply)
+ directory_permission = "0777"
+ file_permission = "0777"
+ filename = "B"
+ id = (known after apply)
}
# local_file.f8[2] will be created
+ resource "local_file" "f8" {
+ content = "test"
+ content_base64sha256 = (known after apply)
+ content_base64sha512 = (known after apply)
+ content_md5 = (known after apply)
+ content_sha1 = (known after apply)
+ content_sha256 = (known after apply)
+ content_sha512 = (known after apply)
+ directory_permission = "0777"
+ file_permission = "0777"
+ filename = "C"
+ id = (known after apply)
}
Plan: 3 to add, 0 to change, 0 to destroy.
Changes to Outputs:
+ a1 = [
+ "A",
+ "B",
+ "C",
]
+ a2 = [
+ "ADDRESS",
+ "NAME",
]
+ a3 = [
+ "B",
VM-Terra
2%
0.42 GB / 0.83 GB
0.05 Mb/s
0.03 Mb/s
71 min
KHAJA


2. 20.120.242.231 (KHAJA)
]
+ a3 = [
+ "B",
+ "A",
]
+ a4 = {
+ address = "B"
+ name = "A"
}
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.
Enter a value: yes
local_file.f8[0]: Creating...
local_file.f8[1]: Creating...
local_file.f8[2]: Creating...
local_file.f8[0]: Creation complete after 0s [id=a94a8fe5ccb19ba61c4c0873d391e987982fbbd3]
local_file.f8[1]: Creation complete after 0s [id=a94a8fe5ccb19ba61c4c0873d391e987982fbbd3]
local_file.f8[2]: Creation complete after 0s [id=a94a8fe5ccb19ba61c4c0873d391e987982fbbd3]
Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
Outputs:
a1 = [
"A",
"B",
"C",
]
a2 = [
"ADDRESS",
"NAME",
]
VM-Terra
0%
0.42 GB / 0.83 GB
0.01 Mb/s
0.00 Mb/s
72 min
KHAJA
/: 11%
/boot: 8%



Terraform console

The terraform console command provides an interactive shell for evaluating Terraform expressions and testing configurations. It's extremely useful for debugging and experimenting with Terraform code.

1. Launching the Console

terraform console

- This opens an interactive prompt where you can evaluate expressions.

2. Basic Usage Examples

A. Evaluating Simple Expressions

terraform

> 1 + 2

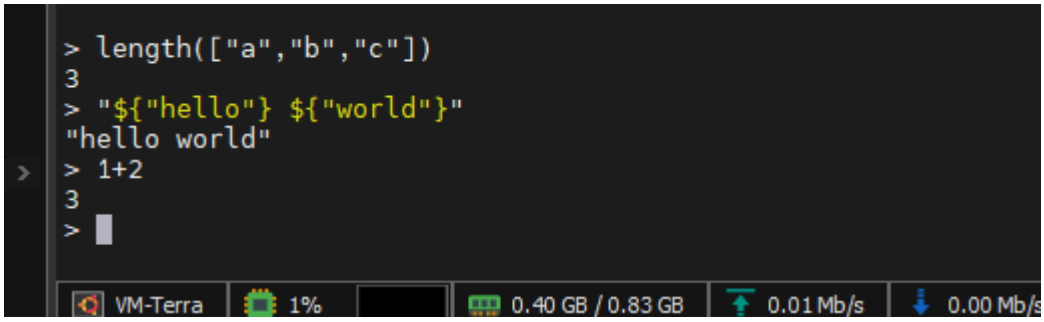
3

> "${"hello"} ${"world"}"

"hello world"

> length(["a", "b", "c"])

3



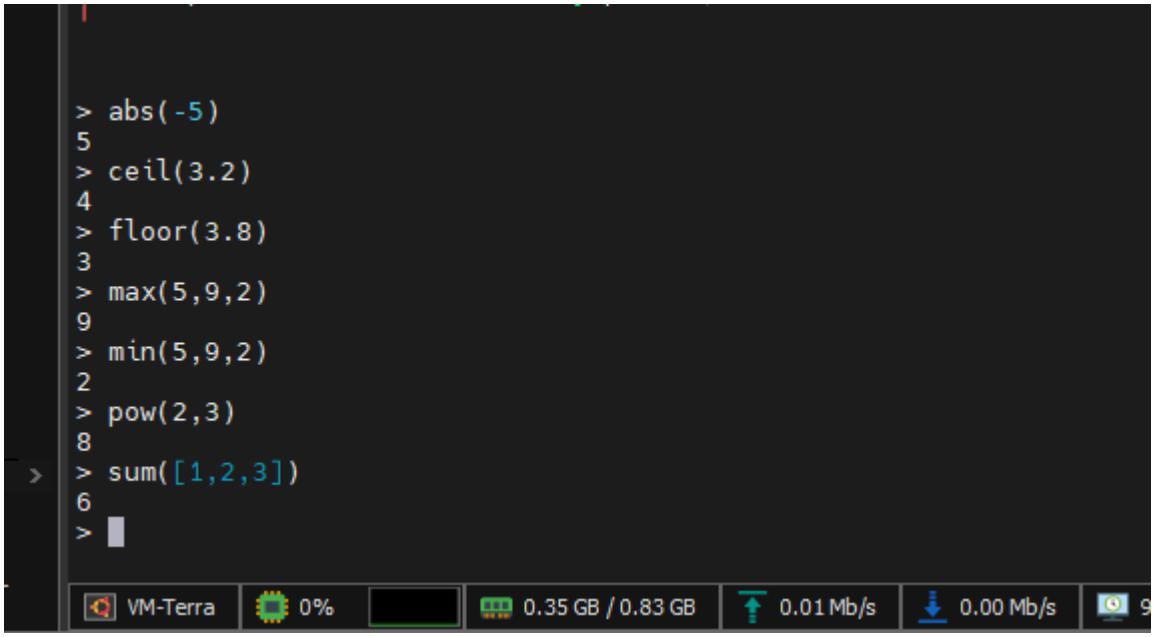
Terraform functions

Terraform provides built-in functions for transforming and combining values. Here's a comprehensive guide to all function categories with practical examples:

- additional capabilities to run/ execute in a smoother way

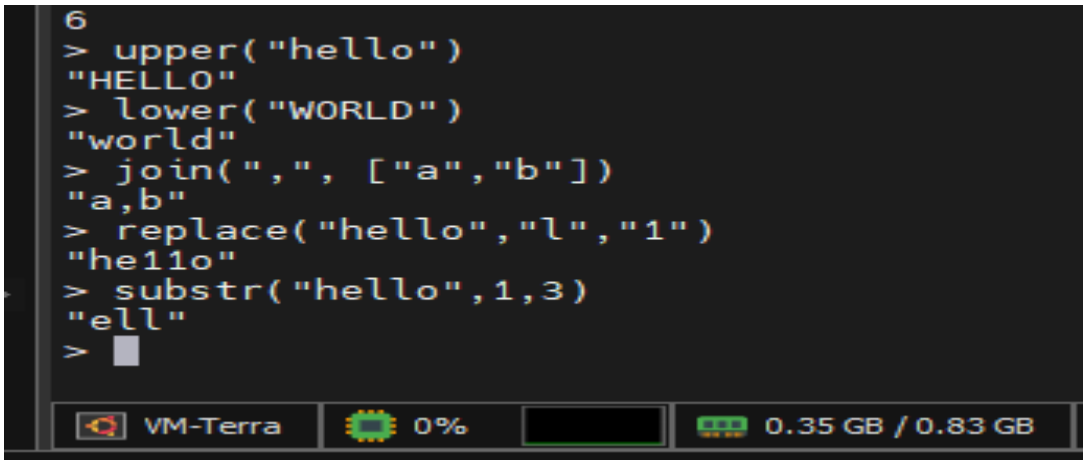
1. Numeric Functions

Function	Definition	Example
`abs(x)`	Absolute value	`abs(-5) → 5`
`ceil(x)`	Rounds up to nearest integer	`ceil(3.2) → 4`
`floor(x)`	Rounds down to nearest integer	`floor(3.8) → 3`
`max(N1,N2,...)`	Returns highest value	`max(5,9,2) → 9`
`min(N1,N2,...)`	Returns lowest value	`min(5,9,2) → 2`
`pow(x,y)`	Raises x to power y	`pow(2,3) → 8`
`sum(list)`	Sums all elements	`sum([1,2,3]) → 6`



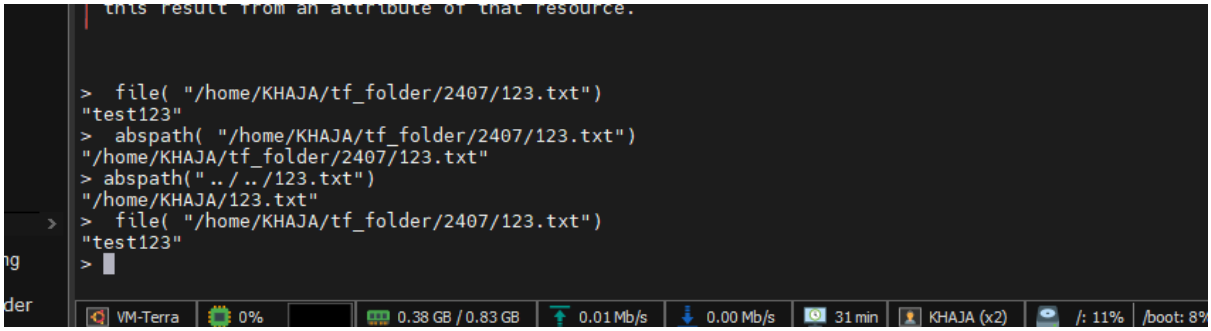
2. String Functions

Function	Definition	Example
`upper(str)`	Converts to uppercase	`upper("hello") → "HELLO"`
`lower(str)`	Converts to lowercase	`lower("WORLD") → "world"`
`join(delim, list)`	Combines list with delimiter	`join(",", ["a","b"]) → "a,b"`
`replace(str,sub,new)`	Replaces substring	`replace("hello","l","1") → "he11o"`
`substr(str,start,len)`	Extracts substring	`substr("hello",1,3) → "ell"`



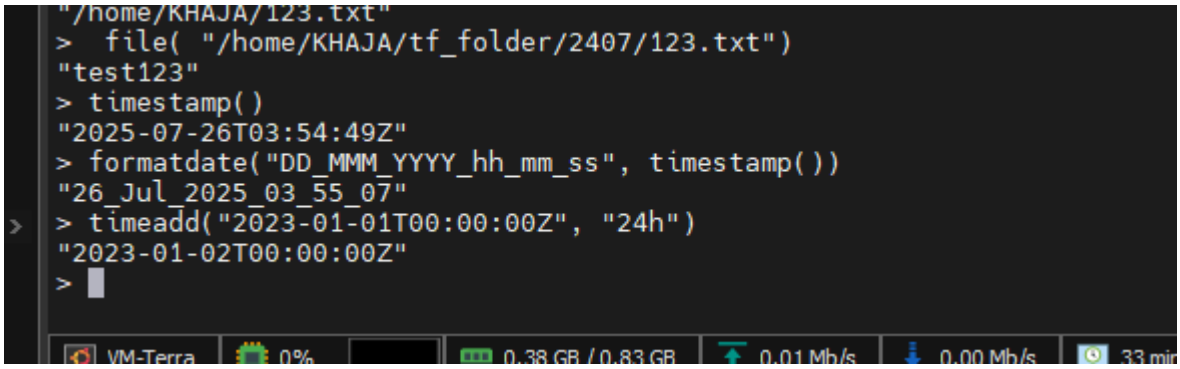
3. Filesystem Functions

Function	Definition	Example
`abspath(path)`	Converts to absolute path	`abspath("./file") → "/home/project/file"`
`file(path)`	Reads file contents	`file("../../nginx.txt"→ file contents`
`fileexists(path)`	Checks if file exists	`fileexists("main.tf") → true`



4. Date/Time Functions

Function	Definition	Example
`timestamp()`	Current UTC timestamp	`timestamp() → "2023-05-15T12:34:56Z"`
`formatdate(fmt,time)`	Formats timestamp	`formatdate("DD-MM-YYYY", timestamp()) → "15-05-2023"`
`timeadd(time,duration)`	Adds time duration	`timeadd("2023-01-01T00:00:00Z", "24h") → adds 1 day`



5. Collection Functions

Function	Definition	Example
`length(collection)`	Count of elements	`length(["a","b"]) → 2`
`toset(list)`	Converts to unique set	`toset(["a","b","a"]) → ["a","b"]`
`sort(list)`	Alphabetical sort	`sort(["b","a"]) → ["a","b"]`
`lookup(map,key,default)`	Safe map lookup	`lookup({a=1},"a",0) → 1`
`contains(list,value)`	Membership check	`contains(["a","b"],"a") → true`
`distinct(list)`	Removes duplicates	`distinct([1,2,2]) → [1,2]`
`concat(lists...)`	Combines lists	`concat([1],[2]) → [1,2]`
`sum(list)`	Numeric sum	`sum([1,2,3]) → 6`